PARALLEL MULTIPROCESSING FOR THE FAST FOURIER TRANSFORM WITH PIPELINE ARCHITECTURE

Field of Invention

The present invention relates to the field of Fast Fourier Transform analysis. In particular, the present invention relates to a parallel processing architecture adapted for use in a pipelined Fast Fourier Transform method and apparatus.

Background of the invention

Physical parameters such as light, sound, temperature, velocity and the like are converted to electrical signals by sensors. An electrical signal may be represented in the time domain as a variable that changes with time. Alternatively, a signal may be represented in the frequency domain as energy at specific frequencies. In the time domain, a sampled data digital signal is a series of data points corresponding to the original physical parameter. In the frequency domain, a sampled data digital signal is represented in the form of a plurality of discrete frequency components such as sine waves. A sampled data signal is transformed from the time domain to the frequency domain by the use of the Discrete Fourier Transform (DFT). Conversely, a sampled data signal is transformed back from the frequency domain into the time domain by the use of the Inverse Discrete Fourier Transform (IDFT).

The Discrete Fourier Transform is a fundamental digital signal-processing transformation used in many applications. Frequency analysis provides spectral information about signals that are further examined or used in further processing. The DFT and IDFT permit a signal

to be processed in the frequency domain. For example, frequency domain processing allows for the efficient computation of the convolution integral useful in linear filtering and for signal correlation analysis. Since the direct computation of the DFT requires a large number of arithmetic operations, the direct computation of the DFT is typically not used in real time applications.

Over the past few decades, a group of algorithms collectively known as Fast Fourier Transform (FFT) have found use in diverse applications, such as digital filtering, audio processing and spectral analysis for speech recognition. The FFT reduces the computational burden so that it may be used for real-time signal processing. In addition, the fields of applications for FFT analysis are continually expanding.

Computational burden

Computation burden is a measure of the number of calculations required by an algorithm. The DFT process starts with a number of input data points and computes a number of output data points. For example, an 8-point DFT may have an 8-point output. See figure 1A. The DFT function is a sum of products, i.e., multiplications to form product terms followed by the addition of product terms to accumulate a sum of products (multiply-accumulate, or MAC operations). See equation (1) below. The direct computation of the DFT requires a large number of such multiply-accumulate mathematical operations, especially as the number of input points is made larger. Multiplications by the twiddle factors W_N^r dominate the arithmetic workload.

To reduce the computational burden imposed by the computationally intensive DFT, previous researchers developed the Fast Fourier Transform (FFT) algorithms in which the number of required mathematical operations is reduced. In one class of FFT methods, the computational burden is reduced based on the divide-and-conquer approach. The principle of the divide-and-conquer approach method is that a large problem is divided into smaller sub-problems that are easier to solve. In the FFT case, the division into sub-problems means that the input data are divided in subsets for which the DFT is computed to form partial DFTs. Then the DFT of the initial data is reconstructed from the partial DFTs. See N. W. Cooley and J. W. Tukey, " An algorithm for machine calculation of complex Fourier series", Math.Comput., Vol. 19 pp. 297-301, Apr 1965. There are two approaches to dividing (also called decimating) the larger calculation task into smaller calculation sub-tasks: decimation in frequency (DIF) and decimation in time (DIT).

Butterfly implementation of the DFT

For example, an 8-point DFT can be divided into four 2-point partial DFTs. The basic 2-point partial DFT is calculated in a computational element called a radix-2 DIT butterfly (or butterfly-computing element) as represented in figure 2A1. Similarly to the DIT butterfly-computing element, figure 2A2 shows the function of a radix-2 DIF butterfly. A radix-2 butterfly has 2 inputs and 2 outputs, and computes a 2-point DFT. Figure 2B shows an FFT using 12 radix-2 butterflies to compute an 8-point DFT. Butterfly-computing elements are arranged in stages. There are three stages 1302, 1304 and 1306 of butterfly calculation. Data, x_n is fed to the input of the butterfly-computing elements in the first stage 1302. After the first stage 1302 of butterfly-computation is complete, the result is fed to the in input of the next stage(s) of butterfly-computing element(s) and so on.

In particular, four radix-2 butterflies operate in parallel in the first stage 1302 to compute 8 partial DFTs. The 8 outputs of the first stage 1302 are combined in 2 additional stages 1304, 1306 to form a complete 8-point DFT output, X_n. Specifically, the second stage 1304 of 4 radix-2 butterflies and the third stage 1306 of 4 radix-2 butterflies comprise a two stage combination phase in which 8 radix-2 butterflies responsive to 8 partial DFTs form the final 8-point DFT function, X_n.

Figure 2C shows an FFT using 32 radix-2 butterflies to compute a 16-point DFT. There are 4 stages of butterfly calculation. Eight radix-2 butterflies operate in parallel in the first stage 1402 where 2-point partial DFTs are calculated. The outputs of the first stage are combined in 3 additional combination stages 1403, 1404 and 1406 to form a complete 16-point DFT output. The output of the second stage 1403 of 8 radix-2 butterflies is coupled to a third stage 1404 of 8 radix-2 butterflies. The output of the third stage 1404 of 8 radix-2 butterflies is coupled to a fourth stage 1406 of 8 radix-2 butterflies, the output of which the final 16-point DFT function. The combination phases 1403, 1404, 1406 comprise a combination phase in which 24 radix-2 butterflies responsive to 16 partial DFTs (from the first phase 1402) form the final 16 point DFT function, X_n.

Higher order butterflies may be used. See figure 2D, which uses 8 radix-4 butterflies in 2 stages 1502, 1502 to compute a 16-point DFT. In general, a radix-r butterfly is a computing element that has r input points and calculates a partial DFT of r output points. In figure 2D, four radix-4 butterflies compute 16 partial DFTs in a first stage 1502. The combination phase 1504 comprises four radix-4 butterflies responsive to 16 partial DFTs (from the first phase 1502) to form the final 16 point DFT function, X_n.

Communication burden

A computational problem involving a large number of calculations may be performed one calculation at a time by using a single computing element. While such a solution uses a minimum of hardware, the time required to complete the calculation may be excessive. To speed up the calculation, a number of computing elements may be used in parallel to perform all or some of the calculations simultaneously. A massively parallel computation will tend to require an excessively large number of parallel-computing elements. Even so, parallel computation is limited by the communication burden. For example, a large number of data and constants may have to be retrieved from memory over a finite capacity data bus. In addition, intermediate results in one parallel-computing element may have to be communicated to another parallel-computing element. The communication burden of an algorithm is a measure of the amount of data that must be moved, and the number of calculations that must be performed in sequence (i.e., that cannot be performed in parallel).

In particular, in a butterfly implementation of the DFT, some of the butterfly calculations cannot be performed simultaneously, i.e., in parallel. Subsequent stages of butterflies cannot begin calculations until earlier stages of butterflies have completed prior calculations. Also, the connections between butterflies in each stage to butterflies in the other stages impose a heavy communication burden between the butterfly computation stages. Thus, parallel implementations of the butterfly DFT are hampered by a heavy communication burden between butterflies.

Summary of the invention

The heavy communication burden between the butterfly stages in the prior art results from structuring the butterfly implementation such that the first butterfly stage computes partial DFTs over the input data, and the latter butterfly stages combine the partial DFTs. In accordance with the present invention, partial DFTs are computed in a plurality of separate parallel processors and then combined in a single stage of combination for the parallel processing algorithm. Also, In accordance with the present invention for the multi-stage parallel processing algorithm, partial DFTs are computed in a plurality of separate parallel circuit boards that contain plurality of separate parallel chips, which contain plurality of separate parallel processors. The output data is obtained by combining firstly the outputs of the plurality of separate parallel processors, secondly by combining the outputs of the plurality of separate parallel chips and finally by combining the plurality of the outputs of the separate parallel circuit boards.

The present architecture is a reorganization of the butterfly calculation of the DFT so as to reduce the communication burden between butterflies implemented in parallel computing elements. In particular, no communication is required between pluralities of separate parallel processors, chips or circuit boards. A combination stage of butterfly calculation is provided which combines the outputs of all the parallel processors (chips or circuit boards).

In accordance with the present invention, the input data points of an N point DFT are divided into subsets of data. A plurality of processors, chips or circuit boards operate in parallel and independently of each other, and each being responsive to each respective subset of the input data points. The partial DFTs at the output of the plurality of parallel processors in each chip are then combined in a single combination phase to provide the

complete partial DFT for a single chip. The partial DFTs at the output of the plurality of parallel chips in each circuit board are then combined in a single combination phase to provide the complete partial DFT for a single board. The partial DFTs at the output of the plurality of parallel circuit boards are then combined in a single combination phase to provide the DFT solution.

In the general case, each output data point of the DFT is a function of all of the input data points. However, by dividing the input data set into subsets, and operating each parallel processor independently of the other parallel processors in accordance with the present invention, the communication between the parallel processors is eliminated, thereby reducing the communication burden. Each or several parallel processors may then be implemented on a separate semiconductor chip or circuit board, without requiring any communication with any of the other parallel processors.

In a second embodiment of the present invention, a radix-r butterfly implementation is provided in which the plurality of independent processors are operated in parallel using the same instructions and accessing the same necessary set of multiplier coefficients from memory at the same time. The resulting algorithm, in which a number of parallel processors operate simultaneously by a single instruction sequence, reduces both the computational burden and the communication burden.

Brief description of the drawings

Figure 1A is a block diagram representation of an 8-point DFT function.

Figure 1B is a block diagram representation of an 8-point DFT obtained by combining two 4-point DFTs.

Figure 1C is a block diagram representation of an 8-point DFT obtained by combining four 2-point DFTs.

Figure 2A1 is a schematic representation of a DIT radix-2 butterfly computation.

Figure 2A2 is a schematic representation of a DIF radix-2 butterfly computation.

Figure 2B is a signal flow graph illustrating an FFT calculation using 12 radix-2 butterflies to compute an 8-point FFT in three stages.

Figure 2C is a signal flow graph illustrating an FFT calculation using 32 radix-2 butterflies to compute a 16-point FFT in 4 stages.

Figure 2D is a signal flow graph illustrating an FFT calculation using 8 radix-4 butterflies to compute a 16-point FFT in two stages.

Figure 3A is a schematic diagram of a DIF butterfly-processing element for an FFT algorithm for use with the present invention.

Figure 3B is a shorthand representation of a DIF butterfly-processing element for use with the present invention.

Figure 4A is a schematic diagram of a DIT butterfly-processing element for an FFT algorithm for use with the present invention.

Figure 4B is a shorthand representation of a DIT butterfly-processing element for use with the present invention.

Figure 5A is a schematic representation of a DIF butterfly for use with the present invention.

Figure 5B is a schematic representation of a DIT butterfly for use with the present invention.

Figure 6A is a schematic representation of a radix-8 DIT butterfly-processing element for use with the present invention.

Figure 6B is a schematic representation of a radix-16 DIF butterfly-processing element for use with the present invention.

Figure 7 illustrates the design methodology for dividing sets of input data into subsets for use in conjunction with the present invention.

Figure 8 is a block diagram of a parallel FFT multiprocessor in accordance with the present invention.

Figure 9 is a block diagram of a generalized alternate embodiment of a parallel FFT multiprocessor in accordance with the present invention.

Figure 10 is a block diagram of a 16-point radix-2 embodiment of a parallel FFT multiprocessor in accordance with the present invention using four 4-point parallel processors.

Figure 11 is a block diagram of a 16-point radix-2 embodiment of a parallel FFT multiprocessor in accordance with the present invention using two 8-point parallel processors.

Detailed description

The definition of the DFT is shown in equation (1), $x_{(n)}$ is the input sequence, $X_{(k)}$ is the output sequence, N is the transform length and w_N is the Nth root of unity ($w_N = e^{-j2\pi/N}$). Both $x_{(n)}$ and $X_{(k)}$ are complex valued number sequences.

$$X_{(k)} = \sum_{n=0}^{n=N-1} x_{(n)} w_N^{nk} , k \in [0, N-1] \quad (1).$$

From equation (1) it can be seen that the computational complexity of the DFT increases as the square of the transform length, and thus, becomes expensive for large N. A class of FFT methods, known as fast algorithms for DFT computation, is based on the divide-and-conquer approach. The principle of the divide-and-conquer approach method is that a large

problem is divided into smaller sub-problems that are easier to solve. In the FFT case, the division into sub-problems means that the input data x_n are divided in subsets on which the DFT is computed. Then the DFT of the initial data is reconstructed from these intermediate results. If this strategy is applied recursively to the intermediate DFTs, an FFT algorithm is obtained. Some of these methods are the Common Factor Algorithm, the Split-Radix Algorithm, the Prime Factor Algorithm and the Winograd Fourier Transform Algorithm (WFTA). Of particular interest is the in place (or butterfly) computation.

The main objective of many FFT algorithms is a reduction in computation and particularly the reduction in the amount of multiplication.

PARALLEL PROCESSING: Dividing the input data into subsets

The communication burden is best addressed at the architecture level where the representation of the DFT in terms of its partial DFTs is restructured mathematically. The problem resides in finding a mathematical model of the combination phase. The problem is to find a way that $X = T_N x$ (a vector with n components) can be recovered from r vectors that are r times shorter. The first step is to divide x itself. The vector $S = (x_0, x_1, .., x_{n-1})$ is split into r shorter pieces and the subdivision continues in till we obtain subsets in each of which we could find a single vector of length r. For the first subdivision, the original set S_0 (level or layer 0) is subdivided into r subsets:

$$S_{10} = x_m , S_{11} = x_{(m+1)}, ..., \text{and } S_{1(r-1)} = x_{(m + (r-1))} \quad (4).$$

The subsets of level or layer 1 could be expressed as:

$$S_{1j_1} = x_{(r^1 n + j_1)} \qquad (5),$$

for $j_1 = 0, 1,.., r-1$ and $n = 0, 1, .., ((N/r) - 1)$. Each piece of these subsets are divided into

r subsets of length $(N/r^2 - 1)$, therefore, each of the subsets $S_{10}, S_{11}, .. , S_{1(r-1)}$ is subdivided

into r subsets yielding to:

the subset of S_{10} will include:

$$S_{200} = x_{r(m)}, S_{210} = x_{r(m+1)}, ..., S_{2(r-1)0} = x_{r(m+(r-1))} \qquad (6),$$

S_{11} would have on level (layer) 2

$$S_{201} = x_{(r(m)+1)}, S_{211} = x_{(r(m+1)+1)},.., S_{2(r-1)1} = x_{(r(m+(r-1)+1)} \qquad (7),$$

$$\cdot$$
$$\cdot$$

and finally $S_{1(r-1)}$ on level (layer) 2 will contain:

$$S_{20(r-1)} = x_{(r(m)+(r-1))}, S_{21(r-1)} = x_{(r(m+1)+(r-1))}, ...,S_{2(r-1)(r-1)} = x_{(r(m+(r-1))+(r-1))} \qquad (8).$$

The second subdivision could be represented by:

$$S_{2j_1 j_2} = x_{(r(rn+j_2)+j_1)} = x_{(r^2 n + r^1 j_2 + r^0 j_1)} \qquad (9),$$

for $j_1 = j_2 = 0, 1,.., r-1$, and $n = 0, 1, .., ((N/r^2) - 1)$.

In general, the subsets of the ith level (layer) could be obtained by:

$$S_{ij_1....j_i} = x_{(r^i n + r^{(i-1)} j_i + r^{i-2} j_{i-1} + ... + r^0 j_1)} \qquad (10),$$

for $i = 1,.., p$, $j_i = j_{i-1} = ... = 0, 1,.., r-1$, $p = (\log_r (N)/\log_r) - 1$, and $n = 0, 1, .., ((N/r^i) - 1)$.

In each subdivided subsets, the coefficients just go alternately into r vectors of length r, x_0,

x_1, and $x_{(r-1)}$ known as the radix r core calculation (or kernel computation). Then from

those vectors we form

$$X_0 = T_r x_0, X_1 = T_r x_1, ..., \text{and } X_{(r-1)} = T_r x_{(r-1)} \qquad (11).$$

Those are the r multiplications by the smaller matrix T_r. As in the replacement of T_N by T_r, the work has been cut by a factor of r. The central problem is to recover X from the (N/r) points vectors X_0, X_1, ... , $X_{(r-1)}$ and Cooley and Tukey noticed how it could be done for radix-2 by introducing the concept of the butterfly computation. Mathematically speaking, the successive subdivisions of the initial data act as if the one-dimensional transform is expressed by a true r^i - dimensional transform. Figures 7, 8 and 9 illustrate the division of the input data set into subsets.

PARALLEL PROCESSING: Finding the mathematical model for the combination phase

Although there is a substantial gain obtained by decomposing the input data into radix r core calculations, as shown in figures 2C and 2D, a severe problem arises in the complexity of the communication load. For higher radices, the communication load will increase drastically with the increasing amount of data. However, the combination phases (where partial DFTs are combined) have certain regularity in their structures. The combination of r vectors in each subdivided subset of length β ($\beta = N/r^i$) is obtained by feeding words of length r into the input of the PE. The kth word at the ith level (layer) is given by:

$$X_{(k, i)} = (X_{(0, k)}, X_{(1, k)}, .., X_{((r-1), k)}) \quad (2),$$

where $k = 0,..,\beta - 1$, $i = 1,..,p$ and $p = (\log_r (N)/\log_r) - 1$.

The combined output is a vector of length $r\beta$, in which the lth output of the kth word at the $(i - 1)^{th}$ level (layer) is stored into the memory address location x_l according to:

$$X_l = lr\beta + k \quad (3),$$

for $l = 0, 1, \ldots\ldots, r - 1$.

The advantage of appropriately breaking the DFT in terms of its partial DFTs is that the number of multiplications and the number of stages may be reduced. The number of stages often corresponds to the amount of global communication and/or memory accesses in implementation, and thus, reduction in the number of stages is beneficial. Minimizing the computational complexity may be done at the algorithmic level of the design process, where the minimization of the communication burden is done at the architectural level.

A fast Fourier transform processor implemented using parallel multiprocessing in accordance with the present invention is shown in figure 8. An input memory 802 is provided to receive the N input data points. There are r parallel processors 807A through 807B, each responsive to 1/r of the input 802 data points driven by the source selection matrix 806, to compute r partial DFTs. Each of the r parallel processors 807A through 807B are implemented in accordance with equation (17) for DIF or equation (19) for DIT. The outputs of the r parallel processors 807A and 807B are driven to the input of the combination phase 817 by the combination reading address generator 816. After the combination phase 817, the output data is driven to the output memory 819 by the combination writing address generator 818. The combination phase 817 is implemented in accordance with equation (37), given below, for DIT or DIF.

Each parallel processor 807A, 807B, comprises a respective reading address generator 808A, 808B, writing address generator 809A, 809B, input memory 810A, 810B, output memory 811A, 811B and an arithmetic processing unit 814A, 814B.

A coefficient memory 804 is provided to store the coefficients that will be needed during the FFT calculations. In particular, a request for a coefficient is made by any one (or all) of the r arithmetic processing unit 814A through 814B. The coefficients needed by each of the arithmetic processing unit 814A, 814B, and the respective internal butterfly computation units are responsive to the same instructions and use the same coefficients 804.

Note that the r parallel processors 806A through 806B, do not need to communicate with each other. Accordingly, there is no communication burden between the r parallel processors 806A through 806B. Each or several parallel processors may be implemented on a separate chip and several chips could be mounted on a circuit board without the need to send or receive data to or from the other parallel processors, chips or circuit boards.

The parallel multiprocessor architecture of the present invention is further generalized in the block diagram of figure 9. An input memory 902 is provided to receive the N input data points. A plurality of parallel processors 906A, 906B, 906C, 906D, 906E, 906F is provided.

Each parallel processor 906A, 906B, 906C, 906D, 906E, 906F includes an arithmetic unit, input and output memories, reading and writing address generators, and each has the same general structure and function as the parallel processor 807A, 807B in the block diagram of

figure 8. Each parallel processor 906A, 906B, 906C, 906D, 906E, 906F is responsive to a subset of the N input data points. The outputs of the parallel processors 906A, 906B, 906C, 906D, 906E and 906F are combined in respective combination phases 907A, 907B, 907C, 907D. Combination phase 907 is equivalent to the combination phase 817 in figure 8 with different parameters. The output of the combination phase 907A, 907B is stored in the input memories of the chips 908A, 908B, 908C, 908D, 908E, 908F (where each chip is an electronic layout forming r parallel processors). The data from the chips 908A, 908B, 908C, 908D, 908E and 908F is combined in combination phases 909A, 909B, 909C, 909D, 909E, 909F and stored in the input memories of the circuit boards 912A, 912B, 912C, 912D, 912E, 912F. A circuit board is an electronic device on which r parallel chips are mounted. The data of the circuit boards 912A, 912B, 912C, 912D, 912E and 912F are combined in combination phases 913A, 913B and stored in the input memories 914A, 914B of the sub frames 904A, 904B (where a sub-frame is an electronic component on which r parallel circuit boards are mounted). The data of the sub frames 904A and 904B, is combined in combination phase 916 and stored in the output memory of main-frame 918. It is noted that the further division of the subset of input data need not be divided in the same ratio as the original division.

A specific embodiment of a 16- point radix-2 DFT utilizing the parallel processing architecture of the present invention is shown in figure 10. Four separate and independent parallel processors 100A, 100B. 100C, 100D are responsive to the input data points. Parallel processor 100A comprises adders 110, 112, 114, 116, 124, 126, 128, 130, and multipliers 118, 120. Adders 110, 114 and multiplier 118 form a radix-2 butterfly 122.

The outputs of the four parallel processors are combined using four radix-4 butterflies 102, 104, 106, 108 to form the output data points.

An alternate embodiment of a 16- point radix-2 DFT utilizing the parallel processing architecture of the present invention is shown in figure 11. Two separate and independent parallel processors 150A, 150B are responsive to the input data points. Parallel processor 150A comprises a plurality of adders and multipliers. In particular, adders 152, 154 and multiplier 156 form a radix-2 butterfly. Similarly, adders 162, 164 and multiplier 166 form a radix-2 butterfly. The outputs of the two parallel processors 150A, 150B are combined using eight radix-2 butterflies formed by adders170 and multipliers 172 to form the output data points.

THE JABER PRODUCT ($\widehat{*}$): Representing the DFT in terms of its partial DFTs.

For a given $r \times r$ square matrix T_r and for a given set of r column vectors of length $X_{0_{(k)}}, X_{1_{(k)}}, \ldots\ldots, X_{(r-1)_{(k)}}$, the Jaber product is defined as the operator $\widehat{*}$ as the following operation:

$$T_r \,\hat{*}\, \begin{bmatrix} X_{0_{(k)}} \\ X_{1_{(k)}} \\ -- \\ X_{(r-1)_{(k)}} \end{bmatrix} = \begin{bmatrix} X_{0_{(0)}} T_{00} + X_{1_{(0)}} T_{01} + \dots\dots + X_{(r-1)_{(0)}} T_{0(r-1)} \\ X_{0_{(1)}} T_{00} + X_{1_{(1)}} T_{01} + \dots\dots + X_{(r-1)_{(1)}} T_{0(r-1)} \\ ------------------- \\ X_{0_{(\beta-1)}} T_{00} + \dots\dots\dots + X_{(r-1)_{(\beta-1)}} T_{0(r-1)} \\ X_{0_{(0)}} T_{10} + X_{1_{(0)}} T_{11} + \dots\dots + X_{(r-1)_{(0)}} T_{1(r-1)} \\ ------------------ \\ X_{0_{(\beta-1)}} T_{10} + \dots\dots\dots + X_{(r-1)_{(\beta-1)}} T_{1(r-1)} \\ ------------------ \\ X_{0_{(0)}} T_{(r-1)0} + \dots\dots\dots + X_{(r-1)_{(0)}} T_{(r-1)(r-1)} \\ ------------------ \\ X_{0_{(\beta-1)}} T_{(r-1)0} + \dots\dots + X_{(r-1)_{(\beta-1)}} T_{(r-1)(r-1)} \end{bmatrix} \quad (12),$$

which is equal to:

$$T_r \,\hat{*}\, \begin{bmatrix} X_{0_{(k)}} \\ X_{1_{(k)}} \\ -- \\ X_{(r-1)_{(k)}} \end{bmatrix} = \begin{bmatrix} X_{0_{(k)}} T_{00} + \dots + X_{(r-1)_{(k)}} T_{0(r-1)} \\ X_{0_{(k)}} T_{10} + \dots + X_{(r-1)_{(k)}} T_{1(r-1)} \\ --------------- \\ X_{0_{(k)}} T_{(r-1)0} + \dots + X_{(r-1)_{(k)}} T_{(r-1)(r-1)} \end{bmatrix} \quad \text{for } k = 0,1,\dots,\beta-1. \quad (13),$$

For simplicity the definition of Jaber product expressed in above two equations could be represented as:

$$T_r \,\hat{*}\, \left[X_{m_{(k)}} \right] = \left[\sum_{m=0}^{r-1} X_{m_{(k)}} T_{(l,m)} \right] \quad \text{for } k = 0,1,\dots,\beta-1 \text{ and } l = 0,1,\dots,r-1 \quad (14).$$

Mathematical representation of the DFT in terms of its partial DFTs.

The operation of a radix r butterfly PE is defined by:

$$X = B_r \times x \quad (15),$$

where $x = [x_0, x_1, \dots\dots, x_{r-1}]^T$ is the input vector and $X = [X_0, X_1, \dots\dots, X_{r-1}]^T$ is the output vector, B_r is the $r \times r$ butterfly matrix, which can be expressed as:

$$B_r = W_{(r,k,i)} \times T_r = \left[B_{r\,DIF(l,m)(k,i)} \right] \quad (16),$$

with

$$B_{r\,DIF(l,m)(k,i)} = w^{((l\,m\,N/r + \tilde{N}\,(k/r^i\,)l\,r^i\,))_N} \quad (17),$$

for the decimation in frequency process,

and

$$B_r = T_r \times W_{(r,k,i)} = \left[B_{r\,DIT(l,m)(k,i)} \right] \quad (18).$$

in which

$$B_{r\,DIT(l,m)(k,1)} = w^{((l\,m\,N/r + \tilde{N}\,(k/r^{(n-1)})m\,r^{(n-i)}))_N} \quad (19),$$

for the decimation in time process. $((x))_N$ denotes x modulo N, $\tilde{N}\,(k\,/\,r^i)$ is defined as the integer part of the division of k and r^i and $W_{(r,\,k,i)}$ represents the twiddle factor matrix expressed by:

$$W_{(r,k,i)} = \begin{bmatrix} w_{(0,k,i)} & 0 & - & 0 \\ 0 & w_{(1,k,1)} & - & 0 \\ - & - & - & - \\ 0 & 0 & - & w_{((r-1),k,i)} \end{bmatrix} = \left[w_{(l,m)(k,1)} \right] \quad (20),$$

$$\text{with } w_{(l,m)(k,i)} = w^{((\tilde{N}(\frac{k}{r^i})l\,r^i))_N} \quad (21),$$

for $l = m$, and 0 elsewhere, for the decimation in frequency process and

$$w_{(l,m)(k,i)} = w^{((\tilde{N}\,(k/r^{(n-i)})m\,r^{(n-1)}))_N} \quad (22),$$

for $l = m$, and 0 elsewhere, for the decimation in time process.
T_r is an r×r matrix representing the adder-tree in the butterfly, where

$$T_r = \begin{bmatrix} w^0 & w^0 & w^0 & - & w^0 \\ w^0 & w^{N/r} & w^{2N/r} & - & w^{(r-1)N/r} \\ w^0 & w^{2N/r} & w^{4N/r} & - & w^{2(r-1)N/r} \\ - & - & - & - & - \\ w^0 & w^{(r-1)N/r} & & - & w^{(r-1)^2 N/r} \end{bmatrix} = \begin{bmatrix} T_{(l,m)} \end{bmatrix} \quad (23),$$

$$\text{where } T_{(l,m)} = w^{((lm-\frac{N}{r}))_N} \quad (24),$$

for $l = m = 0,\ldots, r - 1$.

Based on the proposition in the previous section, equation 1 may be rewritten as:

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} w_N^{kn} = T_r \circledast \begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{(rn)} w_N^{rnk} \\ \sum_{n=0}^{(N/r)-1} x_{(rn+1)} w_N^{(rn+1)k} \\ ---------- \\ \sum_{n=0}^{(N/r)-1} x_{(rn+r-1)} w_N^{(rn+r-1)k} \end{bmatrix} \quad (25),$$

for $k = 0, 1, \ldots\ldots, (N/r) - 1$, and $n = 0, 1, \ldots, N - 1$.

Since,

$$w_N^{rnk} = w_{N/r}^{nk} \quad (26),$$

equation (25) becomes:

$$X_{(k)} = T_r \circledast \begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{(rn)} w_{N/r}^{nk} \\ \sum_{n=0}^{(N/r)-1} x_{(rn+1)} w_{N/r}^{nk} w_N^{k} \\ ---------- \\ \sum_{n=0}^{(N/r)-1} x_{(rn+(r-1))} w_{N/r}^{nk} w_N^{(r-1)k} \end{bmatrix} = T_r \circledast \begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{(rn)} w_{N/r}^{nk} \\ w_N^{k} \sum_{n=0}^{(N/r)-1} x_{(rn+1)} w_{N/r}^{nk} \\ ---------- \\ w_N^{(r-1)k} \sum_{n=0}^{(N/r)-1} x_{(rn+(r-1))} w_{N/r}^{nk} \end{bmatrix} \quad (27),$$

which for simplicity may be expressed as:

$$X_{(k)} = T_r \; \hat{*} \left[w_N^{j_1 k} \sum_{n=0}^{(N/r)-1} x_{(rn+j_1)} w_{N/r}^{nk} \right] \quad (28),$$

where for simplification in notation the column vector in equation 27 is set equal to:

$$\begin{bmatrix} \sum_{n=0}^{(N/r)-1} x_{(rn)} w_{N/r}^{nk} \\ w_N^k \sum_{n=0}^{(N/r)-1} x_{(rn+1)} w_{N/r}^{nk} \\ \overline{} \\ w_N^{(r-1)k} \sum_{n=0}^{(N/r)-1} x_{(rn+(r-1))} w_{N/r}^{nk} \end{bmatrix} = \left[w_N^{j_1 k} \sum_{n=0}^{(N/r)-1} x_{(rn+j_1)} w_{N/r}^{nk} \right] \text{for } j_1 = 0,...,(r-1) \quad (29).$$

If $X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} w_N^{nk}$ represents the DFT of a sequence of length N,

$$X_{p0_{(k)}} = \sum_{n=0}^{(N/r)-1} x_{rn} w_{N/r}^{nk}, X_{p1_{(k)}} = \sum_{n=0}^{(N/r)-1} x_{rn+1} w_{N/r}^{nk},....,X_{p(r-1)_{(k)}} = \sum_{n=0}^{(N/r)-1} x_{rn+(r-1)} w_{N/r}^{nk}$$

represent the DFT of sequences of length N/r. As a result, equation (28) can be expressed as:

$$X_{(k)} = T_r \; \hat{*} \begin{bmatrix} w^0 X_{p0_{(k)}} \\ w_N^k X_{p1_{(k)}} \\ \text{---------} \\ w_N^{(r-1)k} X_{p(r-1)_{(k)}} \end{bmatrix} = T_r \; \hat{*} \left[w_N^{j_1 k} X_{pj_{1(k)}} \right] \quad (30),$$

for $j_1 = 0, 1, ..., r-1$.

Finally equation (30) can be developed according to the definition of Jaber product to yield the following expression:

$$X_{(k)} = \begin{bmatrix} w^0 X_{p0_{(k)}} + w^{0+k} X_{p1_{(k)}} + \ldots\ldots\ldots + w^{0+(r-1)k} X_{p(r-1)_{(k)}} \\ w^0 X_{p0_{(k)}} + w^{N/r+k} X_{p1_{(k)}} + \ldots + w^{(r-1)N/r+(r-1)k} X_{p(r-1)_{(k)}} \\ \text{-------------------------} \\ w^0 X_{p0_{(k)}} + \ldots\ldots\ldots\ldots + w^{(r-1)^2 N/r+(r-1)k} X_{p(r-1)_{(k)}} \end{bmatrix} \quad (31).$$

for $k = 0, 1, \ldots, \beta - 1$.

Due to the fact that:

$$X_{(k)} = T_r \, \hat{*} \, \left\{ \begin{bmatrix} w_N^0 & 0 & - & 0 \\ 0 & w_N^k & - & 0 \\ - & - & - & - \\ 0 & 0 & - & w_N^{(r-1)k} \end{bmatrix} \times \begin{bmatrix} X_{p0_{(k)}} \\ X_{p1_{(k)}} \\ - \\ X_{p(r-1)_{(k)}} \end{bmatrix} \right\} = [T_r \times W_{(k)}] \hat{*} [X_{pj_{1(k)}}] \quad (32),$$

$$\text{where } W_{(k)} = [\, w_{(l,j_1)} \,] \quad (33)$$

$$\text{in which } w_{(l,j_1)} = w^{((j_1 k))_N} \quad (34),$$

for $l = j_1$ and 0 elsewhere, equation 30 in case where $X_{pj_{1(k)}}$ represnts the partial DIF or DIT DFT of the sequence $\displaystyle\sum_{n=0}^{(N/r)-1} x_{(rn+j_1)} \, w_{N/r}^{nk}$ may be expressed by:

$$X_{(k)} = [B_{r \,(\text{Combination DIT})}] \hat{*} [X_{pj_{1(k)}}] \quad (35),$$

for the decimation in time process, and

$$X_{(k)} = [B_{r \,(\text{Combination DIF})}] \hat{*} [X_{pj_{1(k)}}] \quad (36),$$

where $X_{pj_{1(k)}}$ represnts the partial DIT or DIF DFT of the sequence $\displaystyle\sum_{n=0}^{(N/r)-1} x_{(rn+j_1)} \, w_{N/r}^{nk}$.

Further simplification results when the lth transform of the kth word is represented by:

$$X_{l(k)} = \sum_{j_1=0}^{r-1} X_{pj_{1(k)}} \, w_N^{((l j_1 N/r + j_1 k))_N} \quad (37)$$

for $l = 0, 1,\ldots, r - 1, k = 0, 1, \ldots, \beta - 1$, where $((x))_N$ denotes x modulo N.

The lth transform is stored at the address memory location given by equation (4).

PARALLEL MULTIPROCESSING ALGORITHM - Figure 8

The conceptual key of parallel processing is in the reformulation of the Discrete Fourier Transform in terms of its partials DFTs that can be combined in a single combination phase. The present reformulation enables the design of a multiprocessor parallel structure that utilizes r-processing elements in parallel to implement each of the partial DFT computations. For a multiprocessor environment, the parallel processor implementation results in a decrease in time-delay for the complete FFT of $O(r)$. The time delay could be decreased by a factor of $O(r^2)$ by implementing the butterflies with BPEs as described in the appendix hereto, which butterfly computing elements utilizes r complex multipliers in parallel to implement each of the butterfly computations.

A second aspect of the combined parallel multiprocessor and BPE in the appendix is that that the combination is especially adapted for parallel multiprocessing environments. If each butterfly is implemented using the disclosed PBE, each of the r parallel processors would always be executing the same instruction simultaneously, which is very desirable for implementation.

The Bit Reversed Input Ordered Output Parallel Multiprocessing Algorithm

The data are fed into the memory blocks of each processing element on level (layer) 1 by the address generator given in equation (21) $S_{1j_1} = x_{(r^1 n + j_1)}$, for $j_1 = 0, 1,.., r -1$ and n = 0, 1, .., $((N/r) - 1)$.

By denoting $X_{p0_{(k)}}, X_{p1_{(k)}}, ..., X_{p(r-1)_{(k)}}$ as the partial DIF or DIT DFTs of

$$\sum_{n=0}^{(N/r)-1} x_{rn} w_{N/r}^{nk}, \quad \sum_{n=0}^{(N/r)-1} x_{rn+1} w_{N/r}^{nk}, \quad,... \text{ and } \quad \sum_{n=0}^{(N/r)-1} x_{rn+(r-1)} w_{N/r}^{nk} \quad \text{respectively, words}$$

of length r are fed into the input of the parallel multiprocessing processing element (MPE). The writing address generator w_r stores the 1th MPE's output $X_{l_{(k)}}$ of the kth word, generated by equation (37) above into the input memory of level (layer) 0 at the address location:

$$W_r = 1 (N/r) + k \qquad (39),$$

for k = 0, 1, ..., $(N/r^2) - 1$.

MULTISTAGE PARALLEL MULTIPROCESSING - figure 9

Further decomposition of the data in terms of its partial DFTs leads to a multistage parallel multiprocessing as shown in figure 9. This type of implementation is feasible for applications in which a huge amount of data needs to be processed and where rapidity, precision and better resolution are required.

By adopting the following notation:

$$X_{(k)} = T_{r_0} \hat{*} \left[\begin{array}{c} T_{r_{r_0}} \hat{*} \left[W_N^{r_{r_0} j_{r_0} k} \sum_{n=0}^{(N/r_0 r_{r_0})-1} X_{(r_0(r_{r_0}n + j_{r_0}))} W_{N/r_0 r_{r_0}}^{nk} \right] \\ -\; -\; -\; -\; -\; -\; -\; - \\ W_N^{(r_0-1)k} T_{r_{(r_0-1)}} \hat{*} \left[W_N^{r_{(r_0-1)} j_{r_{(r_0-1)}} k} \sum_{n=0}^{(N/r_0 r_{(r_0-1)})-1} X_{(r_0(r_{(r_0-1)}n + j_{r_{(r_0-1)}})+(r_0-1))} W_{N/r_0 r_{(r-1)}}^{nk} \right] \end{array} \right]$$

(40),

which may be represented for simplification in notation by:

$$= T_{r_0} \hat{*} \left[W_N^{j_{r_0} k} T_{r_{r_{j0}}} \hat{*} \left[W_N^{r_{r_{j0}} j_{r_{j0}} k} \left[\sum_{n=0}^{(N/r_0 r_{r_{j0}})-1} X_{(r(r_{r_{j0}}n + j_{r_{r_{j0}}})+j_{r_0})} W_{N/r_0 r_{r_{j0}}}^{nk} \right] \right] \right]$$ (41),

$$= T_{r_0} \hat{*} \left[W_N^{j_{r_0} k} T_{r_{r_{j0}}} \hat{*} \left[W_{N/r_{r_{j0}}}^{j_{r_{j0}} k} \left[\sum_{n=0}^{(N/r_0 r_{r_{j0}})-1} X_{(r(r_{r_{j0}}n + j_{r_{r_{j0}}})+j_{r_0})} W_{N/r_0 r_{r_{j0}}}^{nk} \right] \right] \right]$$ (42),

for $j_{r_0} = 0, 1, \ldots\ldots, r_0 - 1$, and $j_{r_{r_{j0}}} = 0, 1, \ldots\ldots, r_{(r_{j0})} - 1$.

In general the radix r FFT expressed in terms of its ith partial FFTs will be as follow:

$$X_{(k)} = T_r \hat{*} \left[W_N^{j_1 k} T_r \hat{*} \left[W_{N/r}^{j_2 k} \left[\ldots T_r \hat{*} \left[W_{N/r^{(i-1)}}^{j_i k} \left[\sum_{n=0}^{(N/r^i)-1} X_{(r^i n + r^{(i-1)} j_i + \ldots + r^0 j_1)} W_{N/r^i}^{nk} \right] \right] \right] \right] \ldots \right]$$ (43),

for $j_1 = \ldots\ldots = j_i = 0, 1, \ldots\ldots, r - 1$ and $i = 1, \ldots\ldots, p$.

The conceptual key of multistage parallel processing is the formulation of the radix-r FFT as composed of partial FFTs with identical structures and a systematic means of accessing the same multipliers coefficients. The latter architecture enables the design of a multi-layer processing element (MPE), in which each layer utilizes an r-processing element (PE) in parallel to implement each partial DFT computation. In other words, the multi-layer

process acts as if the one-dimensional transform is expressed by a true r^i - dimensional

transform, which is feasible in neural networks computing.

The Radix r Multistage Parallel Multiprocessing Algorithm.

$X_{(K)}$, elements of layer 0 are obtained by combining the $r - 1$ subsets of layer one. Each

subset of layer one is obtained by combining $r - 1$ subsets of layer 2 (see tree model of

figure 7). For the layer i, which contains r^i PE implemented in parallel with r complex

multipliers implemented in parallel as shown in figure 9, (N/r^i) elements of data are fed

into the memory blocks of each processing element by the address generator given in

equation (10) $x_{ij_1\ldots j_i} = x_{(r^i n + r^{(i-1)} j_i + r^{(i-2)} j_{(1-2)} + \ldots + r^0 j_1)}$,

for $j_1 = \ldots\ldots\ldots = j_i = 0, 1, \ldots\ldots, r - 1$.

At this level, r^i partials FFTs are performed in parallel in absence of any

communication between these parallel processing elements (i.e., are independent of each

other) and where they will execute the same instruction simultaneously.

By denoting $X_{ij_1\ldots j_{i(0)}}, X_{ij_0\ldots j_{i(1)}}, \ldots\ldots, X_{ij_0\ldots j_{i(r-1)}}$ the DIT or DIF partials

FFTs of the subsets on level i, therefore the subsets on level (layer) i – 1 are obtained by:

$$X_{(i-1)j_1\ldots j_{(i-1)}(k)} = T_r \hat{*} \left[w_{N/r^{(i-1)}}^{j_i k} X_{ij_0\ldots j_{i(k)}} \right] = T_r \hat{*} \left[w_N^{r^{(i-1)} j_i k} X_{ij_0\ldots j_{i(k)}} \right] \quad (44),$$

$$X_{(i-1)j_1\ldots j_{(i-1)}(k)} = T_r \hat{*} \left[\begin{bmatrix} w_N^0 & 0 & - & 0 \\ 0 & w_N^{r^{(i-1)}k} & - & 0 \\ - & - & - & - \\ 0 & - & - & w_N^{r^{(i-1)}(r-1)k} \end{bmatrix} \times \begin{bmatrix} X_{ij_1\ldots j_{i(0)}} \\ X_{ij_1\ldots j_{i(1)}} \\ - \\ X_{ij_1\ldots j_{i(r-1)}} \end{bmatrix} \right] \quad (45),$$

$$X_{(i-1)j_1 \cdots j_{(i-1)(k)}} = \left[T_r \times W_{(k,i)} \right] \hat{*} \left[X_{ij_1 \cdots j_{i(k)}} \right] \quad (46),$$

$$\text{where } W_{(k,i)} = [\, w_{(l,j_i)} \,] \quad (47),$$

$$\text{in which} \quad w_{(l,j_i)} = w_N^{((r^{(i-1)}j_i k))_N} \quad (48),$$

for $l = j_i$ and 0 elsewhere,

$$\text{and finally } X_{(i-1)j_1 \cdots j_{(i-1)(k)}} = \left[B_{r \, \text{Combination}} \right] \hat{*} \left[X_{ij_1 \cdots j_{i(k)}} \right] \quad (49),$$

for $j_0 = 0$, $j_1 = \cdots = j_i = 0, 1, \ldots, r-1$ $i = p, p-1, \ldots, 1$ and $l = 0, 1, \ldots, r-1$.

Further simplification could results when the lth transform of the kth word at the ith stage is represented by:

$$X_{(i-1)j_1 \cdots j_{(i-1)(l,k,i)}} = \sum_{j_i=0}^{r-1} X_{ij_1 \cdots j_{i(k,i)}} \, w_N^{((l j_i N/r + r^{(i-1)} j_i k))_N} \quad (50),$$

and the lth output of the kth word at the ith stage will be stored into the memory block of its specific PE of the layer $j_{(i-1)}$ at the address location given by:

$$X_{(i-1)j_1 \cdots j_{(i-1)(l,k,i)}} = l\left(\frac{N}{r^i}\right) + k \quad (51),$$

for $k = 0, 1, \ldots, (N/r^{(i+1)}) - 1$.

The subdivision of the input data continues till a radix r core calculation is obtained, which will generate a radix r FFT algorithm.

APPENDIX

Preferred radix-r butterfly -processing element (BPE)

Any radix-r butterfly may be used with the present parallel multiprocessing architecture. However the preferred radix-r butterfly is assembled from a plurality of butterfly processing elements (BPEs). Each BPE, or engine, has a regular structure that can be utilized in an array of butterfly-processing elements (BPEs) each having substantially identical structure.

Mathematical basis of the butterfly implementation

A mathematical term that is a function of r input points and provides a single output point is the basis for the design of the present BPE. To provide the insight forming the basis of the present BPE, the basic DFT equation is factored to group the variables used in multiplication (and simultaneously accessed from memory) into one matrix. In particular, starting from the basic DFT equations, the adder matrix is combined with the twiddle matrix to form a single phase of calculation. By grouping all the multiply calculations into one calculation phase and all the addition calculations into the remaining calculation phases, the total number of calculations is reduced and the degree of parallelism is increased.

For a radix-r DIF butterfly, r identical BPEs are arranged in parallel. Each of the r identical BPEs are substantially identical to each other and are operated in parallel using the same instructions and accessing the necessary set of multiplier coefficients from memory at the same time. The outputs of the r identical BPEs form the DFT's r output points.

For a radix-r DIT butterfly, (r-1) identical BPEs are arranged in parallel. Each of the (r-1) identical BPEs is substantially identical to the others and operates in parallel using the same instructions and accessing the necessary set of multiplier constants from memory at the same time. The outputs of the (r-1) identical BPEs form the DFT as (r-1) of the r output points of the butterfly. The remaining output point (X_0) of the DFT is formed as the sum of the r input points.

Trivial multiplications encountered during the execution of particular butterflies may be avoided by simple checks on the coefficient addresses. Avoiding trivial multiplications reduces the computational load of particular butterflies.

An FFT implementation is composed of a plurality of radix-r butterflies with identical BPEs and a systematic addressing scheme for accessing the corresponding multiplier coefficients. Each radix-r butterfly utilizes the basic computing unit (BPE), with r (or r-1) complex multipliers in parallel to implement each of the butterfly computations' output. There is a simple mapping relationship from the three indices (FFT stage, radix-r butterfly, butterfly-processing element) to the addresses of the needed multiplier coefficients. The simple mapping from the three indices to the addresses of the necessary multiplier coefficients accommodates the complexity of higher order radix and mixed radix butterfly implementations of the DFT. In a multiprocessor environment, much of the calculations are performed in parallel to greatly increase processing speed.

By using the disclosed BPE in the implementation of the radix-r butterfly, an FFT implementation is achieved with a reduced number of calculations and a reduced number

of stages of calculations. In addition, the amount of parallelism, both within the butterfly processing element (BPE) calculation phases and within the overall FFT algorithm butterfly stages permits the use of parallel processing to increase overall FFT calculation speed.

The basic operation of a radix-r butterfly PE is the so-called butterfly in which r inputs are combined to give the r outputs via the operation:

$$X = B_r \times x \qquad (2A),$$

where $x = [x_{(0)}, x_{(1)}, \ldots, x_{(r-1)}]^T$ is the input vector and $X = [\ X_{(0)}, X_{(1)}, \ldots, X_{(r-1)}\ ]^T$ is the output vector.

A special operator, B_r is the r×r butterfly matrix, which can be expressed as

$$B_r = W_N^r \times T_r \qquad (3A)$$

for the decimation in frequency process, and

$$B_r = T_r \times W_N^r \qquad (4A)$$

for the decimation in time process.

$W_N^r = \text{diag}\,(1, w_N^p, w_N^{2p}, \ldots\ldots\ldots\ldots, w_N^{(r-1)p})$ represents the twiddle factor and T_r is an r×r matrix representing the adder-tree in the butterfly, where

$$T_r = \begin{bmatrix} w^0 & w^0 & w^0 & - & w^0 \\ w^0 & w^{N/r} & w^{2N/r} & - & w^{(r-1)N/r} \\ w^0 & w^{2N/r} & w^{4N/r} & - & w^{2(r-1)N/r} \\ - & - & - & - & - \\ w^0 & w^{(r-1)N/r} & \cdot & - & w^{(r-1)^2 N/r} \end{bmatrix} = \left[T_{(l,m)} \right] \qquad (5A),$$

$$\text{where } T_{(l,m)} = w^{((lm\frac{N}{r}))N} \qquad (6A),$$

$l = m = 0,\ldots, r - 1$ and $((x))_N = x$ modulo N.

A higher radix butterfly will decrease the number of multiplications and the number of butterfly stages, which is the motivation for to implement a higher radix butterfly. Since the higher radix reduces the communication load by reducing the number of stages, the remaining problem is reduction of the computational burden. Factoring the adder matrix Tr (radix 4 case) yielding to the following expression reduces the computational load

$$T_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad (7A).$$

As a result, there is a reduction of 4 complex additions for the radix-4 butterfly. For the radix-8 butterfly, there is a reduction of 32 complex additions, 4 real additions and 4 real multiplications. The reduction of calculations is due to the detection of trivial multiplications (i.e., by 1 or -1) and by incorporating (absorbing) the multiplications by j or $-j$ in the addition operation by switching the real and imaginary parts of the data. As indicated, factoring the adder matrix reduces the number of required computations. Although the computation burden is reduced, the complexity of the implementation of such butterflies will increase for higher radices. Also, for higher radices, the number of non-trivial multiplications increases.

An important observation is that both the adder matrix T_r and the twiddle matrix W_N^r, contain twiddle factors. To control the variation of the twiddle factor during the calculation

of a complete FFT, the twiddle factors and the adder matrix are incorporated in a single-phase of calculation.

The special operator B_r is defined according to equation (3A), where B_r is the product of the twiddle factor matrix W_N^r and the adder matrix T_r.

So, by defining $W_{(r, k, i)}$ the set of the twiddle factor matrices W_N^r as:

$$W_{(r,k,i)} = \begin{bmatrix} W_{(0,k,i)} & 0 & - & 0 \\ 0 & W_{(1,k,i)} & - & 0 \\ - & - & - & - \\ 0 & 0 & - & W_{((r-1),k,i)} \end{bmatrix} = \begin{bmatrix} W_{(l,m)_{(k,i)}} \end{bmatrix} \quad (8A),$$

in which,

$$W_{(l,m)(k,i)} = w^{((\tilde{N}(\frac{k}{r^i})l\,r^l))_N} \quad \text{for } l = m, \text{ and } 0 \text{ elsewhere} \quad (9A),$$

therefore, the radix $- r$ computation $B_{r\,\text{DIF}}$ will be expressed:

$$B_{r\,\text{DIF}} = W_{(r, k, i)} \times T_r = \begin{bmatrix} B_{r\,\text{DIF}\,(l,m)_{(k,i)}} \end{bmatrix} \quad (10A),$$

with $B_{r\,\text{DIF}\,(l,m)_{(k,i)}} = w^{((l\,m\,N/r + \tilde{N}\,(k/r^i)l\,r^l))_N} \quad (11A),$

$l = m = 0,\ldots, r-1$, $i = 0,1..,n-1$, $k = 0,1, (N/r) - 1$, $((x))_N$ denotes x modulo N and $\tilde{N}\,(k / r^i)$ is defined as the integer part of the division of k and r^i.

As a result, the operation of a radix-r PE for the DIF FFT will formulated by:

the colon vector $X_{(r,k,i)} = B_{r\,\text{DIF}} \times x = \begin{bmatrix} X_{(l)_{(k,i)}} \end{bmatrix} \quad (12A),$

where the lth output $X_{(l)_{(k,i)}} = \sum_{m=0}^{r-1} x_{(m)} w^{((l\,m\,N/r + \tilde{N}\,(k/r^i)l\,r^l))_N} \quad (13A),$

Using the same reasoning as for the radix-r DIF FFT above, the operation of a radix-r DIT

FFT is derived below. In accordance with equation (4A), B_r is the product of the adder

matrix T_r and the twiddle factor matrix W_N^r, which is equal to:

$$B_{r\ DIT} = T_r \times W_{(r,k,i)} = \left[B_{r\ DIT\ (l,m)_{(k,i)}}\right] \quad (14A),$$

$$\text{in which } B_{r\ DIT\,(l,m)_{(k,i)}} = w^{((l\,m\,N/r+\tilde{N}\,(k/r^{(n-1)})m\,r^{(n-1)}))_N} \quad (15A),$$

$$\text{and } W_{(r,k,i)} = \begin{bmatrix} W_{(0,k,i)} & 0 & - & 0 \\ 0 & W_{(1,k,i)} & - & 0 \\ - & - & - & - \\ 0 & 0 & - & W_{((r-1),k,i)} \end{bmatrix} = \left[W_{(l,m)_{(k,i)}}\right] \quad (16A),$$

where $w_{(l,m)_{(k,i)}} = w^{((\tilde{N}\,(k/r^{(n-1)})m\,r^{(n-i)}))_N}$ for $l = m$, and 0 elsewhere (17A),

and $n = (\log N/\log r) - 1$.

As a result, the operation of a radix $-$ r PE for the DIT FFT will be:

$$\text{the colon vector } X_{(r,k,i)} = B_{r\ DIT} \times x = \left[X_{(l)_{(k,i)}}\right] \quad (18A),$$

$$\text{where the } l^{th} \text{ output } X_{(l)_{(k,i)}} = \sum_{m=0}^{r-1} X_{(m)} w^{((l\,m\,N/r+\tilde{N}\,(k/r^{(n-i)})m\,r^{(n-i)}))_N} \quad (19A).$$

Radix-r butterfly for an FFT algorithm

The conceptual key to forming an FFT algorithm is building radix–r FFT butterflies

composed of BPEs having identical structures and a systematic means of accessing the

corresponding multiplier coefficients stored in memory. The present approach reduces the number of required multipliers and adders.

Each butterfly-processing element utilizes either r complex multipliers in parallel (for a DIF implementation) or $(r - 1)$ complex multipliers in parallel (for a DIT implementation) to implement each of the butterfly computations. There is a simple mapping from the three indices (FFT stage, butterfly, processing element) to the addresses of the multiplier coefficients needed in figures 3A and 4A.

Figures 3A and 3B illustrate a DIF BPE for an FFT. A plurality of r complex multipliers 10A, 10B, 10C, 10D and an adder 14 are provided. One input of each multiplier 10A, 10B, 10C, 10D is coupled to each respective input data point of the input sequence, x. The outputs of each multiplier, 10A, 10B, 10C and 10D are coupled to the inputs of adder 14. The output of adder 14 is coupled to one of the output data points of the output sequence, X. As shown in figure 3B, a generalized butterfly-processing element 12A, 12B, 12C, 12D and 16 is represented by the specialized operator, $B_{r\,DIF}$.

Figures 4A and 4B illustrate a DIT butterfly-processing element (BPE) for an FFT. A plurality of r complex multipliers 20A, 20B, 20C, and an adder 24 are provided. One input of each multiplier 20A, 20B, 20C, is coupled to each respective input data point of the input sequence, x. The outputs of each multiplier, 20A, 20B and 20C are coupled to the inputs of adder 24. The output of adder 24 is coupled to one of the output data points of the output sequence, X. As shown in figure 4B, the specialized operator, $B_{r\,DIT}$, represents a generalized butterfly-processing element 22A, 22B, 22C and 26.

Figures 5A and 5B illustrate how each of the butterfly-processing elements (BPEs) represented by the specialized operators, B_{DIF} and B_{DIT} are arranged to form DIF and DIT butterflies respectively. In particular, figure 5A illustrates a DIF radix-r butterfly. In figure 5A, r butterfly-processing elements 30A, 30B, 30C are arranged in parallel. The inputs of each of the r BPEs 30A, 30B, 30C are respectively coupled to each respective input data point of the input sequence, x. The outputs of each of the r BPEs 30A, 30B, 30C are respectively coupled to each respective output data point of the output sequence, X. Each of the butterfly-processing elements (BPEs) 30A, 30B, 30C, which are operated in parallel, are substantially identical as shown in figure 4B.

Figure 5B illustrates a radix-r DIT butterfly. In figure 5B, r-1 butterfly-processing elements 32A, 32B are arranged in parallel. The inputs of each of the r-1 BPEs 32A, 32B are respectively coupled to each respective input data point of the input sequence, x. The outputs of each of the r-1 BPEs 32A, 32B are respectively coupled to each respective output data point of the output sequence, X, except for X_0. Each of the BPEs 32A, 32B, which are operated in parallel, is substantially identical to the others as shown in figure 3B. Furthermore, an adder 34 is provided to generate the lower order output data point, X_0. The r input terminals of adder 34 are respectively coupled to each respective input data point of the input sequence, x. The output of adder 34 is coupled to the lower order output data point, X_0.

As shown in figures 5A and 5B, the present radix – r FFT butterfly is composed of parallel computing elements. During the FFT calculation, the simultaneous execution of r butterflies in parallel on r separate processors is feasible during each FFT stage. Each the r

parallel processors would always be executing the same instruction simultaneously, which is very desirable for implementation on some of the latest DSP cards. Trivial multiplications encountered during the execution of a particular butterflies (specific radices) may be avoided by simple checks on the coefficients addresses. Avoiding trivial multiplications reduces the computational load of particular butterflies. Trivial multiplications could also be controlled by hardware implementation by using a complex multiplier bypass for $p = 0$, where the trivial multiplication could be computed by two simple multipliers.

Reduction in computation can be achieved, by controlling the trivial multiplication in each word when $((lmN/r + \tilde{N}(k / r^i) \ lr^j))_N$ or $((lmN/r^{(n-i)} + \tilde{N}(k / r^{(n-i)}) \ mr \ r^i))_N$ are equal to a multiple of N/r. As a result, the most significant advantage is that the multiplication by j and -j will be absorbed, yielding to a huge reduction in shifting and negating processes, and the only trivial multiplication left is the multiplication by 1 or -1. In this case the multiplier is by passed whenever the operators $\beta_{(i)}$ satisfies the condition of being a multiple N/r.

Figures 6A and 6B illustrate the regular internal structure of the present radix-r butterfly-processing element (BPE). A radix-8 DIT butterfly-processing element illustrated in figure 6A includes a plurality of 7 (r-1=7, where r = 8) multipliers 48 and adders 49. The multipliers 48 correspond to the (r-1) multipliers 20A, 20B, 20C in figure 4A, and the adders 49 correspond to adder 24 in figure 4A. To compute each output data point, there are 4 phases of computation. In the first phase 40, multipliers 48 multiply each input data point by a coefficient retrieved from memory. Note that all of the coefficients required by

multipliers 48 are retrieved within phase 1. In the next three phases 42, 44, 46, the outputs of the multipliers 48 are summed in adders 49, in pairs. That is, in the second phase 42, four pair of multiplier 48 outputs are added together. In the third phase 44 two pair of adder outputs from the second phase 42 are added together. In the fourth phase 46, one pair of adder outputs from the third phase 44 are added together to form a single output data point.

A radix-16 DIF butterfly-processing element illustrated in figure 6B includes a plurality of 16 multipliers 58 and adders 59. The 16 multipliers 58 correspond to the r multipliers 10A, 10B, 10C, 10D in figure 3A and the adders 59 correspond to adder 24 in figure 3A. To compute each output data point, there are 5 phases of computation. In the first phase 50, multipliers 58 multiply each input data point by a coefficient retrieved from memory. Note that all of the coefficients required by multipliers 58 are retrieved within phase 1. In the next four phases 52, 54, 56, 60, the outputs of the multipliers 58 are summed in adders 59, in pairs. That is, in the second phase 52, eight pair of multiplier 58 outputs are added together. In the third phase 54 four pair of adder outputs from the second phase 52 are added together. In the fourth phase 56, two pair of adder outputs from the third phase 54 are added together. In the fifth phase 60, one pair of adder outputs from the fourth phase 56 are added together to form a single output data point.

As can be seen from figures 6A and 6B, multipliers 48, 58 may be implemented with parallel computing elements because the multiplications are occurring simultaneously in a single phase. Also, due to the regular structure of the DIT and DIF butterfly-processing

elements, an increase in radix, is accommodated by an increase in the number of additional addition phases.

Thus, in applications where minimizing the overall FFT processing time is the most important factor, such as in real time signal processing applications, a parallel implementation of the radix-r butterfly is beneficial.